Escuela de ingeniería en computación
Ingeniería en computación
IC1803 - Taller de programación

Numeradores y denominadores para un cociente dado

Proyecto 1

Aaron Pérez Alfaro
a.perez.1@estudiantec.cr
2024185088

Alajuela, Costa Rica
Abril 2024

1. Resumen ejecutivo

El proyecto consistía en la solución a un problema planteado, el cual se necesitaba encontrar dos números donde el cociente dé como resultado un número introducido por el usuario (C), y a su vez, los resultados tienen que tener la condición de que ambos números pueden tener un mínimo de cuatro dígitos y un máximo de cinco, además, los dos números entre ellos no pueden sobrepasar la cantidad de dígitos repetidos especificados por el usuario, donde R da la cantidad de recurrencias permitidas.

El usuario a la hora de utilizar el programa, debe introducir cuantos casos de prueba va a necesitar, donde T es la cantidad de casos y tiene que ser mayor a 0 y menor o igual a 100. Seguido tendrá que colocar de forma seguida el número que desea encontrar y la cantidad de recurrencias de los dígitos y así la cantidad de casos de prueba que introdujo. Y el resultado se almacena en el archivo output.txt, donde los resultados aparecen de mayor a menor con respecto al numerador, donde cada caso está separado por un salto de línea y si el caso no tiene resultado simplemente deja una línea en blanco.

El algoritmo principal de este problema sería en función del concepto de divisibilidad, ya que, en el programa se deben buscar dos números que divididos dén C. El concepto de divisibilidad utilizado en el código es:

$$a|b \Rightarrow b = ac$$
$$\Rightarrow \frac{b}{c} = a$$

Donde c es el número que introdujo el usuario para encontrar y b es el número a buscar por iteración y si el módulo de b con c es cero, entonces la división daría como resultado a, osea, encontramos los números que divididos dan como resultado C. Seguido, verificamos que los dos números cumplan con el limitante de R, osea cuantas veces se repiten los dígitos de ambos números en conjunto.

En resumen, la solución fue de ir iterando desde el número 10000 hasta 99999; dentro del loop hacemos la división del b con c y verificamos que el resultado sea mayor o igual que 1000, sino, sigue con el siguiente número del ciclo. Si es mayor a 1000 se comprueba que ambos números cumplen con los requisitos y si pasa, inserta una cadena con el formato $numerador/denominador = c$, con un salto de línea en la cadena, así hasta terminar con el ciclo. Y por último se invierte la lista y se combinan todo los elementos en una cadena.

La sección de comprobar los números, sería básicamente, si el denominador es mayor o igual que 1000 y menor a 10000, osea que solamente tiene cuatro dígitos, entonces se le agrega un cero al número. Después se unen los dos números en una lista y se comprueban todos los elementos de la lista, para saber si la cantidad de dígitos no es mayor que R y sus valores de retorno son: true o false, false si sobrepasa R y true si se cumple con la condición. Y por último los resultados se insertan en un archivo de texto llamado output.txt.

Con respecto a los resultados, se utilizó en la gráfica (figura 1) con C = 5 y R = 5, donde se probó de 0 a 200 casos de prueba, mostrando la eficiencia del código para obtener los resultados, donde para el caso de prueba 200 dió como resultado 9 segundos, con el límite de tiempo de 15 segundos. Asimismo, en la carpeta casos de prueba, se encuentran un total de 10 casos donde se prueba el código con diferentes datos de entrada, siendo graficada con la biblioteca *matplotlib* [1] para agilizar la forma de graficar los resultados obtenidos según los casos solicitados.

A lo largo del proyecto surgieron varios inconvenientes, por ejemplo: no se encontraba una forma eficiente de buscar los dos números, ya que, de un principio se trataba de buscar ambos números a la vez, otro problema fue el encontrar una forma de entregar los resultados al usuario, ya que, al inicio se imprimía por consola y eso generaba que el programa aumentara mucho el tiempo de ejecución, por lo cual, se optó por insertar los resultados en un archivo de texto.

2. Introducción

El proyecto consistía en la creación de un programa bajo los principios de *backtracking* [2], el cual encontraba pares de números donde su cociente daba como resultado un número natural específico. Con la limitante de tiempo de 15 segundos de ejecución del programa.

A lo largo de la documentación se explica las funciones y algoritmos utilizados a lo largo del programa, además se evidencia el proposito de cada función implementada, y con varios archivos de casos de prueba con una gráfica mostrando los tiempos de ejecución del programa, donde muestra la eficiencia del código. También se comentan las complecaciones a lo largo del desarrollo del proyecto y su solución a dicho problema. Y por último se muestran los aprendizajes que se obtuvieron mientras se desarrollaba el proyecto.

3. Marco teórico

Para la resolución del problema, utilicé el ide visual studio code, con el lenguaje python3, ya que, visual es muy amigable con su entorno, además, de las facilidades, por la implementación de herramientas como el apartado de extensiones para descargar utilidades de la comunidad y también su sección de control de código, para subir los archivos a github [3]. Y se utilizó python por su entorno fácil de utilizar haciendolo muy sencillo de aprender y empezar con la programación y su buen manejo con números inmensos y su gran productividad, ya que, hace al desarrollador escribir programas con menos líneas de código [4].

Las bibliotecas usadas fueron las de *time* y *Matplotlib*, la primera está implementada en el paquete de python, y su función (en este código) fue de medir el tiempo de ejecución, ya que, era tedioso el estar utilizando la funcionalidad de *linux* o la web *ideone*. Y la segunda, fue utilizada para crear la gráfica para introducirla en la sección de resulados de las pruebas. Estas bibliotecas en la versión de entrega, serán eliminadas, osea solo fueron implementadas en el apartado de desarrollo, para mayor comodidad al desarrollador a la hora de escribir el código.

4. Descripción de la solución

4.1. Detalle del algoritmo

La función principal es, por medio de un while ir sumandole el valor de C al numerador b, e ir probando con cada numerador hasta llegar a 999999 que sería el límite, ya que, después de eso, los números serían de más de cinco dígitos. Y para poder dar los resultados de mayor a menor, cada división encontrada se introducía a una lista, por lo cual, cuando tuviera todos los resultados insertados se invertía la lista y retornaba un string con todos los elementos de la lista, usando join().

El programa consta de dos funciones: *todo y foo* y al final está todo el código de E/S.

- **foo:** foo es la función principal, en el cual solo se reciben dos parámetros: C *y* R, el primero es el número a encontrar por medio del cociente de dos números y el segundo es el total de dıgitos que se pueden repetir.

  Esta función se encarga de por medio de un while ir buscando el numerador, con el cual dividido por C dé el denominador, por medio del concepto de divisibilidad. E introduce en una lista, los resultados a medida que los encuentra por medio de la función *todo*. Y por último, retorna un string de los elementos obtenidos de mayor a menor según el numerador, con el formato de cada elemento $numerador/denominador = C$. Un ejemplo de los valores que retorna *foo* sería: 99776/49888=299774/49887=2. Ya que de por medio de los dos resultados, existe un salto de línea.

- **todo:** se encarga de la verificación de los dígitos de los números, por ejemplo, dependiendo de R, el código permitirá una repetición máxima de dígitos, por numerador y denominador. En resumen de cómo funciona la función es: primero, si el denominador tiene 4 dígitos, por medio

de una multiplicación de 10 se le suma un 0 a los dígitos, ejemplo: 1234 pasa a 12340, ya que, aun que no se vea que el número tenga un 0, en profundidad si lo tiene. Segundo, se unen los dos números en una lista, e iteramos todos los elementos, y haciendo un count(i) para buscar cuantas veces ese dígito se repetía entre el numerador y el denominador.

El código de *E/S* se encarga de obtener por medio de inputs los valores que se introducen por consola. Para el primer input se recibe T, el cual son el número de casos a introducir por medio de un for, el cual permite introducir los valores de C y R juntos, por lo cual, se tiene que usar la función *split()* para separar los elementos insertados y almacernarlos en las variables y esos mismos introducirlos a una lista, donde se tienen todos los valores de C y R introducidos. Luego por medio de un for se iteran los casos y se va llamando la función con los valores, y a su vez se van agregando los resultados a una lista, para luego esa lista iterarla, e ir escribiendo en el archivo de texto.

4.2. Complicaciones

Al principio no encontraba una forma eficiente de buscar los numeros que divididos den C, hasta que encontré el concepto de division, tal que, $a|b \Rightarrow b = aq$ siendo q el C, y por despeje, solo tendríamos que buscar el numerador y dividir por c.

Otra complicación encontrada sería la forma de verificar los dígitos de los números del numerador y denominador, ya que, los algoritmos que utilizaba eran ineficientes, donde se creaban variables innecesarias, loops que podían simplificarse en solo uno, entre otros.

Y además, encontré la ineficiencia de imprimir por consola usando el print(), ya que, para casos pequeños es optimo, pero cuando tiene que escribir por consola grandes cantidades de outputs, hace que el tiempo de ejecución se incremente. Pero es curioso que si imprimimos solo la lista de los resultados, así: [1,2,3,4] es muy rápido, pero cuando tiene que imprimir cada elemento de esa lista, osea: 1, 2, 3, 4; todos en lineas diferentes, se tarda mucho en mostrarlos. Ya que, el código tiene que solicitar permisos de escritura por consola, y eso causa que el tiempo se dispare a mayor cantidad de prints solicitados [5].

5. Resultados de la pruebas

En la carpeta casos de prueba se encontrarán todos los casos de prueba realizados, con su respectivo tiempo de ejecución, sus inputs y resultados. Así mismo se mostrará una gráfica con varias casos prueba.

En esta gráfica (figura 1) se muestra el tiempo de ejecución para los casos de prueba de 1 a 200 para $C = 5$ y $R = 5$. Cabe recalcar que el tiempo de ejecución introducido en la gráfica es aproximado y cada computadora puede presentar resultados de tiempo distintos aproximados.

Esta gráfica fue creada usando la biblioteca *matplotlib* con los valores introducidos manualmente [1].

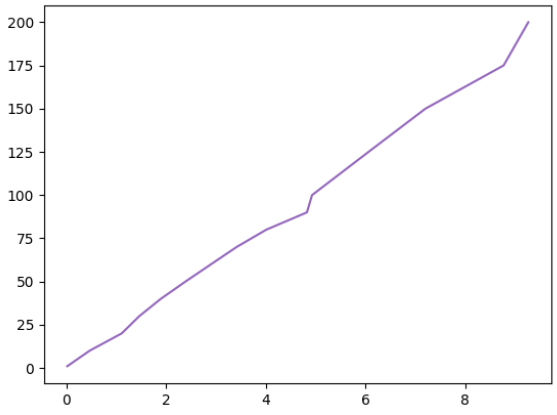

Figura 1: x: segundos, y: casos de prueba

6. Conclusiones

En relación a los resultados obtenidos, introducidos en los casos de prueba, se puede concluir que el tiempo de ejecución para uno o varios C y R, crece a medida que aumenta T, siempre en el peor de los casos, ya que, puede que para un C y R no tengan resultados.

Con respecto al programa, se pudo comprobar la eficacia de los inputs introducidos, arreglando el problema de los prints, intercambiandolo por introducir el output en un archivo de texto, evitando que el tiempo de ejecución aumente por el excesivo uso de los prints.

Otro punto importante, para $C = 1$ y $R = 5$, sería el peor de los casos en los que se pondría a prueba el programa, ya que, para solo un caso de prueba, genera alrededor de 82296 resultados, y para dos casos de prueba, usando ambos $C = 1$ y $R = 5$, generaría 164593 resultados, osea el doble. Por lo tanto, si se quisiera hacer el caso de $T = 100$, para $C = 1$ y $R = 5$ para todos los casos, sería: $82296 \cdot 100 = 8229600$ resultados.

7. Aprendizajes

1. Aprendí un mejor manejo de las listas y métodos en python [6].
2. Aprendí a priorizar la eficiencia y rendimiento del programa.
3. Comprendí lo importante de la utilización de algoritmos obtimos para la creación de la más mínima función, ya que, podría afectar la eficiencia en tiempo de los programas.
4. Aprendí a implementar una función para el control de tiempo de ejecución de los programas [7].
5. Aprendí a formatear los strings en python [8].
6. Aprendí lo ineficiente de usar los prints con grandes listas de elementos.

Referencias

[1] P. Jorge. "Graficar en Python con Matplotlib y NumPy. Numython." (2016), dirección: https://numython.github.io/posts/2016/02/graficar-en-python-con-matplotlib-y/.

[2] S. D. M. Aibin. "Backtracking Algorithms." (2024), dirección: https://www.baeldung.com/cs/backtracking-algorithms#:~:text=Backtracking%20is%20an%20algorithmic%20technique,satisfy%20them%20will%20be%20removed.

[3] F. Flores. "Qué es Visual Studio Code y qué ventajas ofrece." (2022), dirección: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/.

[4] C. P. Santiago De Cuba D. “El lenguaje de programación Python.” (n.d.), dirección: https://www.redalyc.org/pdf/1815/181531232001.pdf.

[5] J. Half. “How to create a new text file using Python. Stack Overflow.” (2024), dirección: https://stackoverflow.com/questions/48959098/how-to-create-a-new-text-file-using-python.

[6] W. schools. “Python List/Array Methods.” (n.d.), dirección: https://www.w3schools.com/python/python_ref_list.asp.

[7] E. libro de python. “Medir tiempo de ejecución.” (n.d.), dirección: https://ellibrodepython.com/tiempo-ejecucion-python.

[8] W. schools. “Python String format() Method.” (n.d.), dirección: https://www.w3schools.com/python/ref_string_format.asp.